Objective-C & iOS Workshop

Patrick Jayet Oliver Gepp Sascha Thoeni

26th Sept. 2013

1 Stack

1.1 Create a New Project

Goal: In this exercise you will create a new XCode-Project, get an overview of the project structure and run a first test.

1. Create a new Xcode-Project Open Xcode 5 and choose from the menubar File > New > Project...
2. Now choose Single View Application and click Next
3. Set the following values as project settings and click Next to finish the project wizard.
4. A new empty single view application is created and all necessary classes to run the app are added to the project. Your project tree should now have at least the following files in it.

   In the folder RpnCalc you can find all the files needed to build a runnable application. All test classes are stored in the folder RpnCalcTest. The wizard already generated an example test class RpnCalcTests.m with a failing test:

   ```
   - (void)testExample{
           XCTFail(@"No implementation for \"%s\"", __PRETTY_FUNCTION__);
   }
   ```

5. To run the all the tests (actually only the one implemented by XCode), click on Product > Test from the menubar. XCode now builds the application, starts the simulator and runs the tests. The Issue Navigator of Xcode will show you now, that the test has failed and provides you also with the failure message of the test.
6. Replace

   ```
   XCTFail(...);
   ```

   with

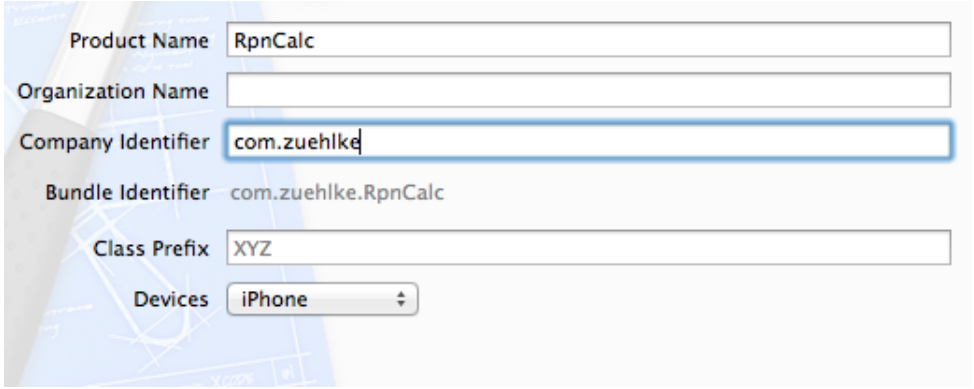

Figure 1: Create new Project

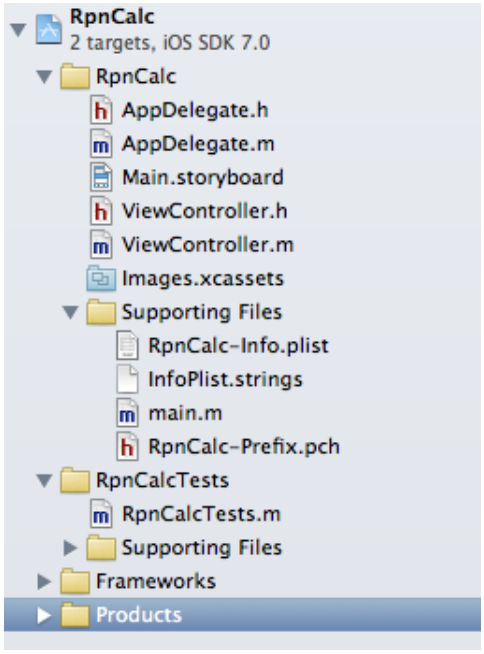

Figure 2: Project files

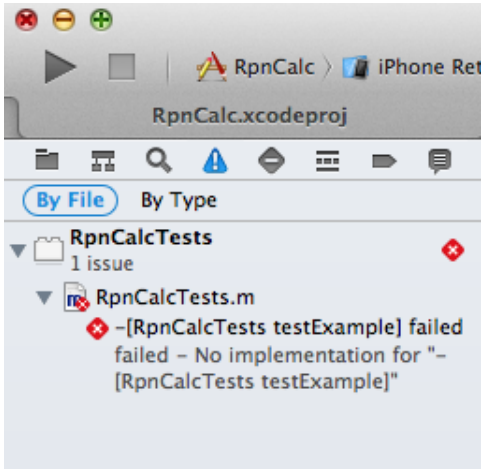

Figure 3: Unit Test

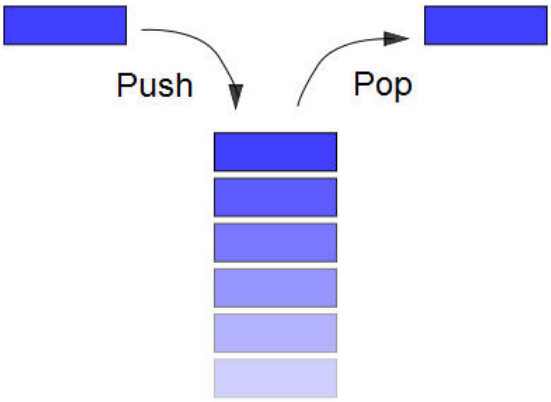

Figure 4: Stack

```
XCTAssert(true);
```

in the testExample method of the RpnCalcTests.m file and run the test again to verify that the test succeeds now.

1.2 Implement a Stack

Goal: In this exercise you will implement your own stack class. It would be used in a later exercise to store the entered numbers of the calculator.

The stack is simply a LIFO (last in first out) queue, which means the last element added to the queue is the first that get returned when polling for the next element.

The tests for the stack class will be given to you, so you can always check if your methods are implemented correctly.

1. Create the stack class
   - Select the folder RpnCalc in the project tree by clicking on it and then choose File > New > File... from the menubar.
   - Choose Objectiv-C Class and click Next.
   - Name the class Stack, click Next and then choose the folder RpnCalc (same hierarchy level as the *.xcodeproj file) and click on Create.
   - Now you should see the new files Stack.m and Stack.h in the project tree.

2. Add the following method declarations to the Stack.h class.

```
- (BOOL) isEmpty;
- (void) push:(id) item;        //add object to stack
- (id) pop;                     //get (last) object from stack
- (NSInteger) size;             //number of objects in stack
- (NSArray*) arrayFromStack;    //return datasource of stack
```

3. Implementing the stack
   - Implement all the (empty) method bodies in the Stack.m class.
   - Declare a mutable array as private property to store the numbers added to the stack by the push method. Put the declaration directly after the import statement in the implementation file.

```
@interface Stack ()

@property (strong) NSMutableArray* array;

@end
```

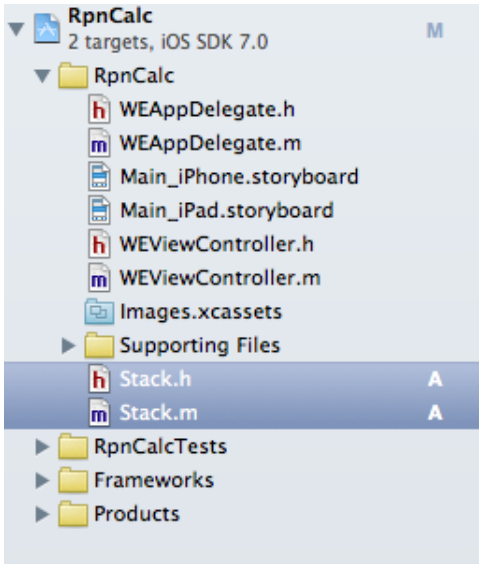

Figure 5: The Stack class

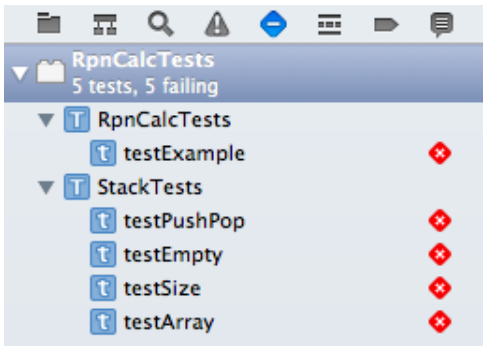

Figure 6: Testing Navigator

- Implement the init method of the class

```
- (id)init  {
        self = [super init];
        if (self) {
                self.array = [[NSMutableArray alloc] init];
        }
        return self;
}
```

4. Download the file StackTests.m in the Exercise1.2 Folder of the github repository

   https://github.com/pajai/RpnCalc

   and Drap&Drop the file into the RpnCalcTests folder in the project tree. Check the option Copy items into destinations group's folder and choose as Target RpnCalcTests.

5. Run the all the tests by clicking on Product > Test in the menubar. Go to the Testing Navigator where you can verify, that all tests failed.

   Notice that you can see all the test failure messages in the Issue Navigator.

6. Now implement the logic of all the methods in Stack.m until all tests are running successfull. Hint: The tests reveal some part of the stack logic.

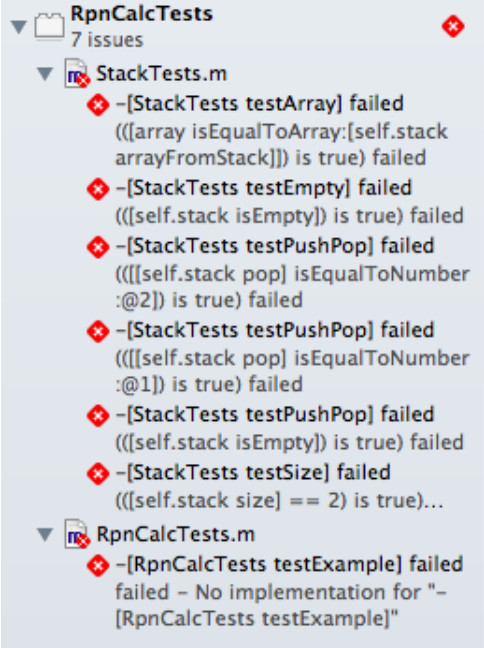

Figure 7: Test failure messages

2 UI

2.1 Linking Interface Items

Goal: In this exercise, we will see how to link items defined in the Interface Builder – classes, outlets and event methods – with the corresponding elements in the code.

1. Open the skeleton project RpnCalcPart2. The setup of the project comprises:
   - The model: a Stack and Computer model classes.
   - A storyboard file where the UI of the app is defined.
   - Various less important files.
2. Create a class CalcViewController, which should subclass UIViewController
   - As seen in the exercise 1.
   - The new class 'CalcViewController' should be a subclass of 'UIViewController'.
   - We have know an empty view controller where some default methods are already implemented
3. Link the controller class in the Interface Builder with the newly created controller class in the code
   - In the Project Navigator, click on the file 'Main.storyboard'. The Interface Builder is shown with the interface file.
   - On the left side of the Interface Builder, in the view hierarchy tree, click on the the file 'View Controller' (see screenshot 8)
   - On the right side select the Identity Inspector (see screenshot 8)
   - Under 'Custom Class', type the name of the newly created controller class 'CalcViewController'.
   - The link between the class in the code and in IB is done.
4. Define an outlet for the result text view

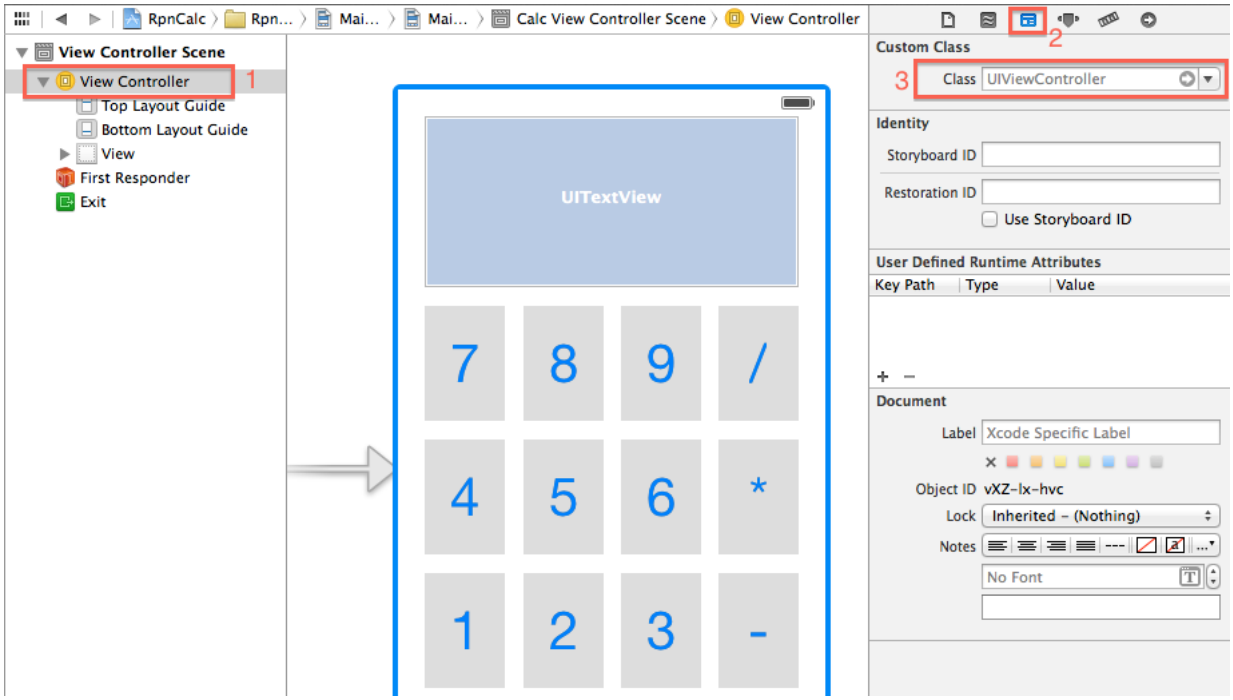

Figure 8: Screenshot of IB showing the Identity Inspector and Custom Class field.

- In the 'CalcViewController' (i.e. in the header .h), add the following property between '@interface' and '@end':

```
@property (strong) IBOutlet UITextView* resultView;
```

- The prefix 'IBOutlet' does nothing in the code, except to indicate to IB that this property will be used as an UI item outlet.
- Go to IB (select 'Main.storyboard').
- Right click on 'Calc View Controller' in the left side, a dark gray panel appears, where we see the name of our result property 'resultView' (see screenshot 9)
- Drag-click from the circle at the right of 'resultView' to the element 'UITextView' on the right (see screenshot 9).
- Now the connection is done between the property in the code and the UI element in IB.

5. In 'CalcViewController', define UI callback methods.

- In order to get events in the code when we press on the calculator keys, we do the connection with the desired events in IB.
- First define in the controller three callback methods:

```
- (IBAction)digitPressed:(id)sender {
    UIButton* button = sender;
    NSString* digit = button.currentTitle;
    NSLog(@"Digit pressed %@", digit);
    // further implementation
}
- (IBAction)operationPressed:(id)sender {
    UIButton* button = sender;
    NSString* operation = button.currentTitle;
    NSLog(@"Operation pressed %@", operation);
    // further implementation
}
```

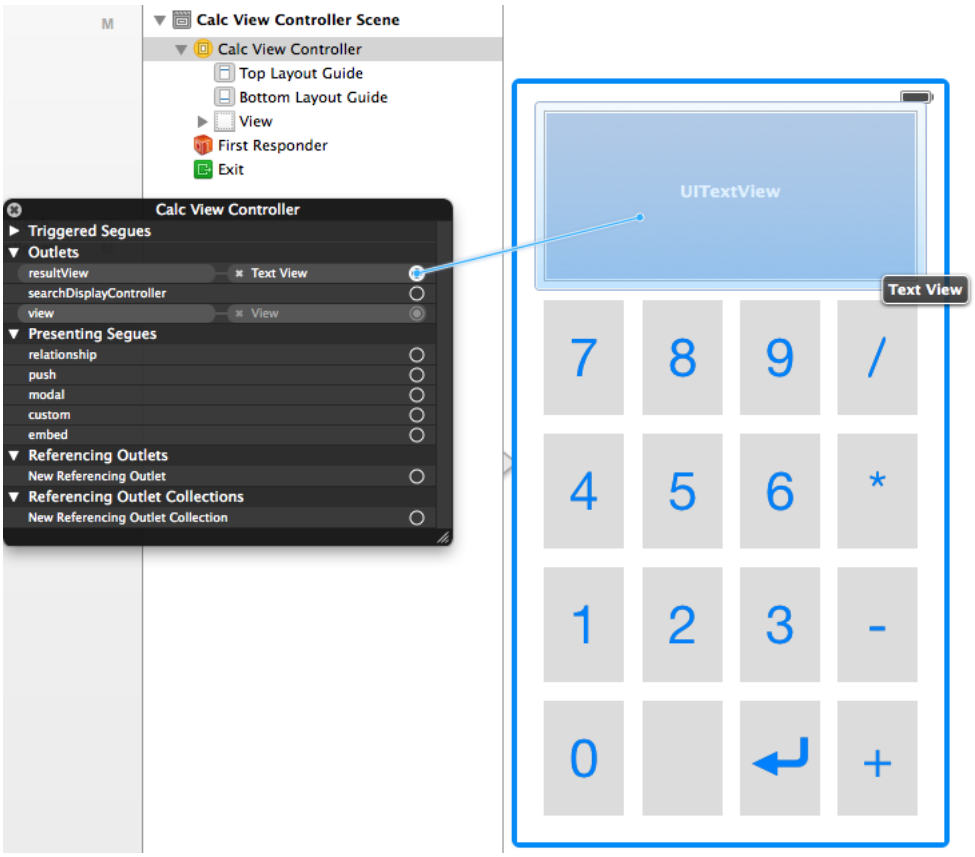

Figure 9: Connecting a view from IB with an outlet in the code.

```
- (IBAction)enterPressed:(id)sender {
    NSLog(@"Enter pressed");
    // further implementation
}
```

- Similarly to the outlet definition, the 'IBAction' keyword does nothing in the code – it is a synonym to 'void' – except to indicate to IB that the corresponding methods are going to receive IB events. The 'sender' parameter is always the view where the event comes from – in this case an UIButton.
- Go to the IB, right click on a digit button, connect the event 'Touch Up Inside' with the controller 'Calc View Controller' on the left and select the method 'digitPressed:'. You have now connected one button. Do the same for the other digit buttons (to the same method).
- Now connect all four operator keys '+', '-', '*' and '/' to the method 'operationPressed:'. Here again the event 'Touch Up Inside'.
- Finally connect the enter key to the method 'enterPressed:'.

6. Now you can start the simulator (the play button on the top left of Xcode). You should see the calculator popping up. When you press on a key, you should see in the console the message of the corresponding callback method.

2.2 Linking the Business Logic

Goal: in this exercise, we will see how to call the business logic and refresh the UI from the callback methods in the controller.

1. In the controller (the .m file) define and initialize a private property

   - According to the following snippet – should be before the '@implementation' block.

     ```
     @interface CalcViewController ()
     @property (readwrite) Computer* computer;
     @end
     ```

   - Remember to import "Computer.h" at the top of the file, for the type 'Computer' to be visible.
   - In the method 'viewDidLoad', initialize the property. A property should always be accessed using the dot notation, e.g. 'self.foo'.

     ```
     self.computer = [[Computer alloc] init];
     ```

2. In the controller (the .m file), define a method to update the UI

   - Use the following snippet

     ```
     - (void)updateUI
     {
         // code here
     }
     ```

   - Complete the missing part. The method should:
     - Read the resulting text from the computer object.
     - Set the text content of the resultView – have a look at the text property of UITextView.

3. Call 'updateUI' from the end of 'viewDidLoad'.

4. Now complete the three callback methods 'digitPressed:', 'operationPressed:' and 'enterPressed:'.

   - Each method should do the following:
     - Call the corresponding method on the 'computer' property.
     - Call the method to update the UI.

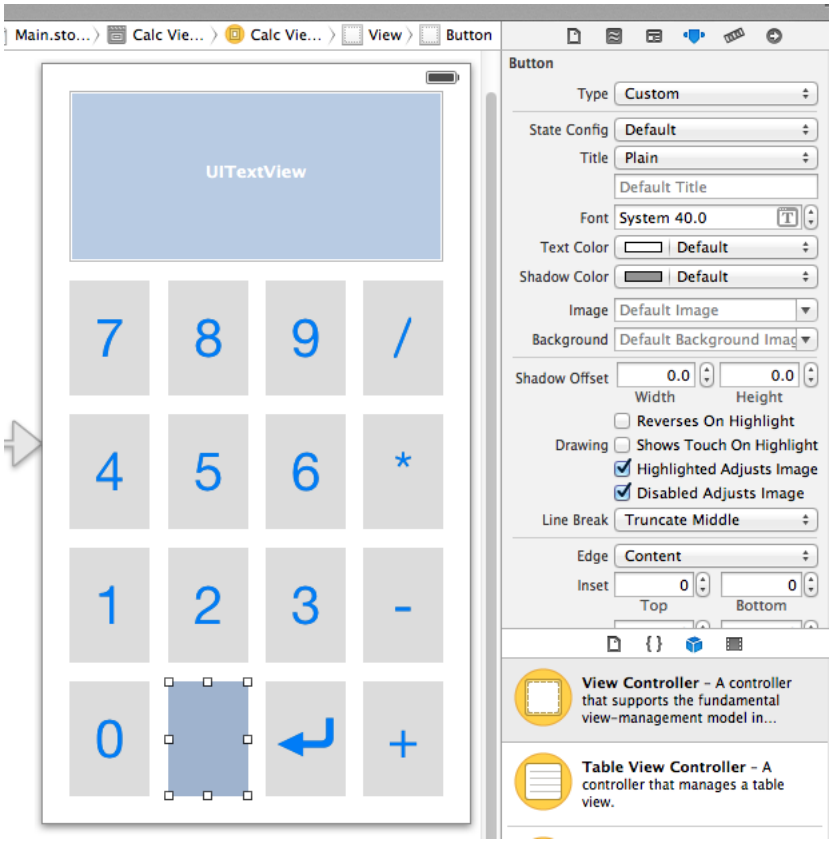

Figure 10: Customizing the play button

5. That's it. Everything should be in place now. Build and run the project. The calculator should work correctly.

   - Hint: this is a calculator which works in RPN mode – reverse polnish notation. To compute '1 + 2' you need the following steps:
     - Press '1'
     - Press 'Enter'
     - Press '2'
     - Press '+'

3 API from an External Framework & Gesture Recognizer

3.1 Say it!

Goal: In this Exercise we will teach our calculator to speak the result. For this we will update the UI and use an external framework for the speech synthesis.

1. Open the storyboard (Main.storyboard) and click the empty button in the lower row.
2. In Attribute Inspector locate the entry "Image" and set its value to loudspeaker by using the dropdown.
3. Switch to Assistant editor perspective an make sure you see the storyboard on the left and your CalcViewController.m file on the right. Drag a line from the loudspeaker button to an empty space in the m-file by dragging ctrl.
4. In the popup define the method name "loudSpeakerPressed".

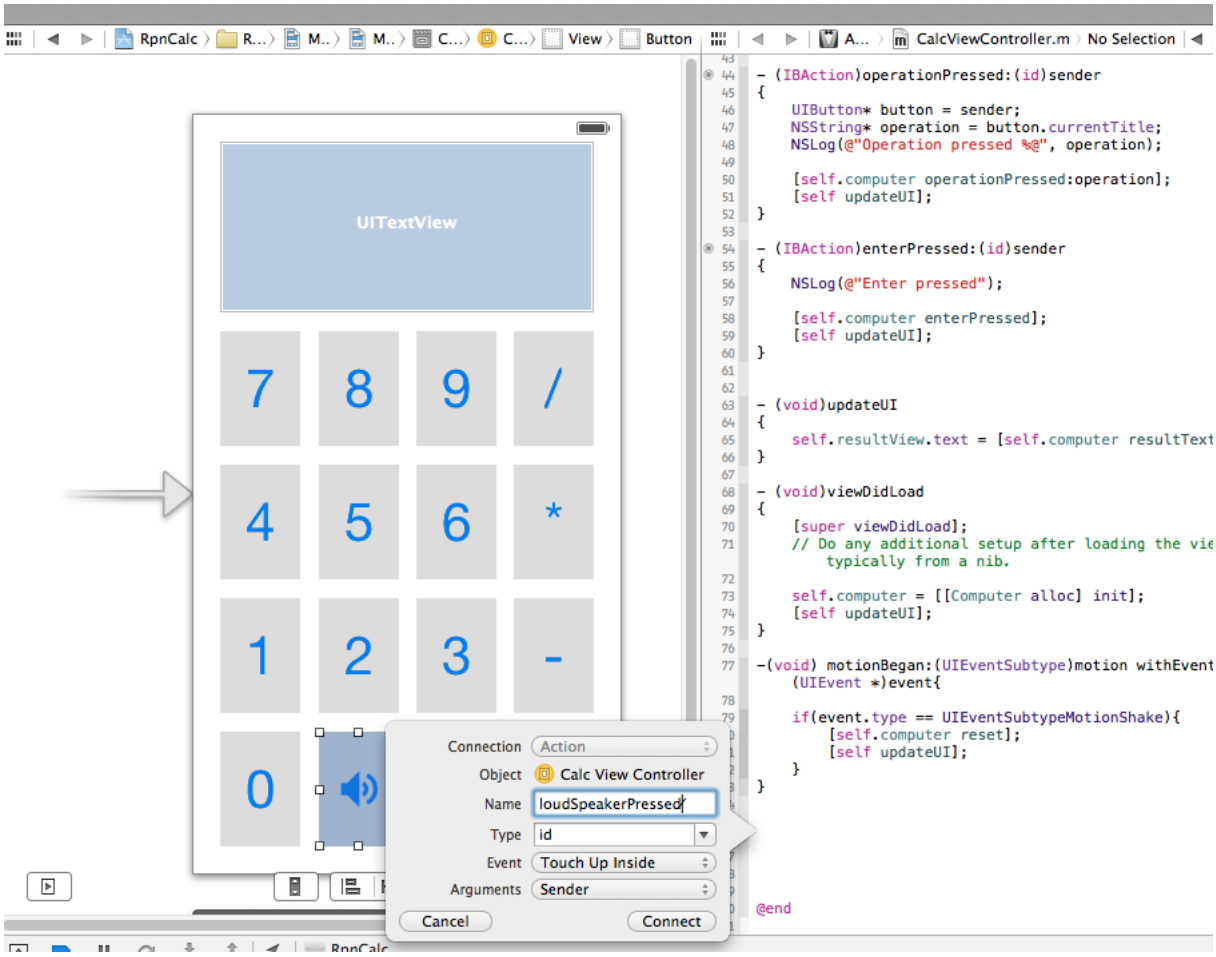

Figure 11: Defining an IBAction with Interface Builder

5. Import the AVFoundation framework by selecting your project root in the Project Navigator. Scroll to the section 'Linked Framework and Libraries' and hit the '+ button'. From the list select 'AVFoundation.framework'. Navigate to your CalcViewController.m file and add import

   ```
   #import <AVFoundation/AVFoundation.h>
   ```

6. Add these lines of code to the loudSpekerPressed method:

   ```
   AVSpeechUtterance *utterance = [AVSpeechUtterance speechUtteranceWithString:''@Hello ''World];
   AVSpeechSynthesizer *synthesizer = [AVSpeechsynthesizer alloc] init];
   [synthesizer speakUtterance:utterance];
   ```

7. Modify the code so that the calculator result is played.

8. Optional: Optimize your code so that utterance and synthesizer is only instantiated once.

9. Optional: Play around with the options of AVSpeechUtterance. How about using a different language or rate? Tip: Have a look in the documentation (Alt+Click on class name).

3.2 Shake it!

Goal: In this exercise we will reset our calculator by shaking the device.

1. Open class CalcViewController.m and add these lines of code:

   ```
   -(void) motionBegann:(UIEventSubtype*) motion withEvent:(UIEvent*)event{
       if(event.type == UIEventSubtypeMotionShake){
           NSLog(''@Shake it''!);
       }
   }
   ```

2. Test your code in the simulator. Simulator can emulate a shake event from the 'Hardware' menu.

3. Modify your method to reset the model and display the result in the calculator display. Tip: You need to modify the classes ViewController.m, Computer.h, Computer.m, Stack.h and Stack.m. Use the method

   ```
   -(void) removeAllObjects;
   ```

   on NSMutableArray to reset the stack.

4. Optional: Play around with your calculator. Be creative – how about a calculator that speaks every number that is pressed?